Entwicklung eines Intel HD Audio Sounkartentreibers in Rust

Präsentation zur Bachelorarbeit

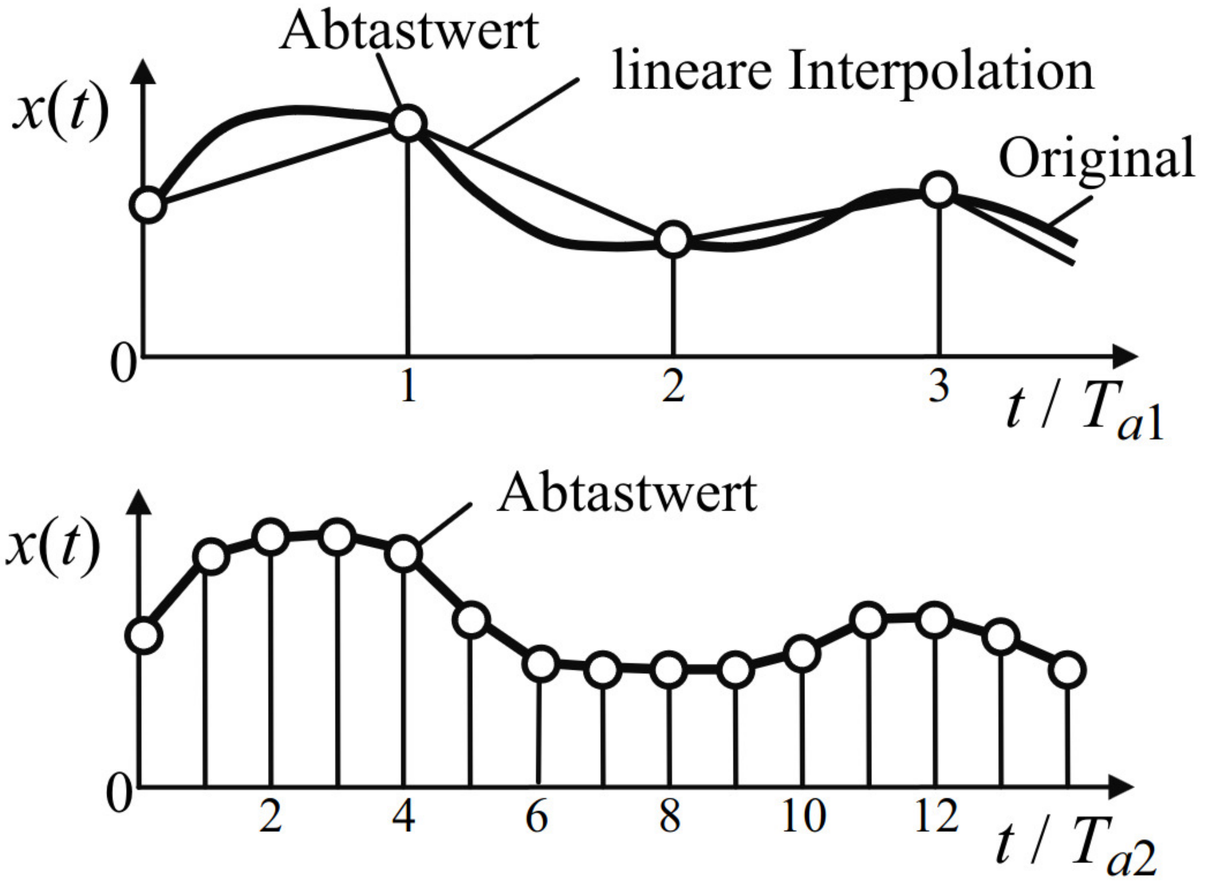
Sebastian Heidelberg, 03.06.2024

Zielsetzung

- Bereitstellung einer API zur Bedienung einer Intel HD Audio Soundkarte durch das D³OS (Rust)
- Wiedergabe eines im Systemspeicher definierten Audiosignals über einen analogen Ausgang der Soundkarte mittels dieser API

PCM

- Pulse-Code-Modulation: Standardverfahren zur Digitalisierung von Audiosignalen
- Umwandlung eines zeitkontinuierlichen Signals (analog) in ein zeitdiskretes (digital)
- DAC/ADC, Sample, Samplerate (z.B. 48kHz), Bittiefe (z.B. 16 Bit)



Nachrichtentechnik, eine Einführung für alle Studiengänge, Vieweg+Teubner, 2006, Bild 3-10, S. 68

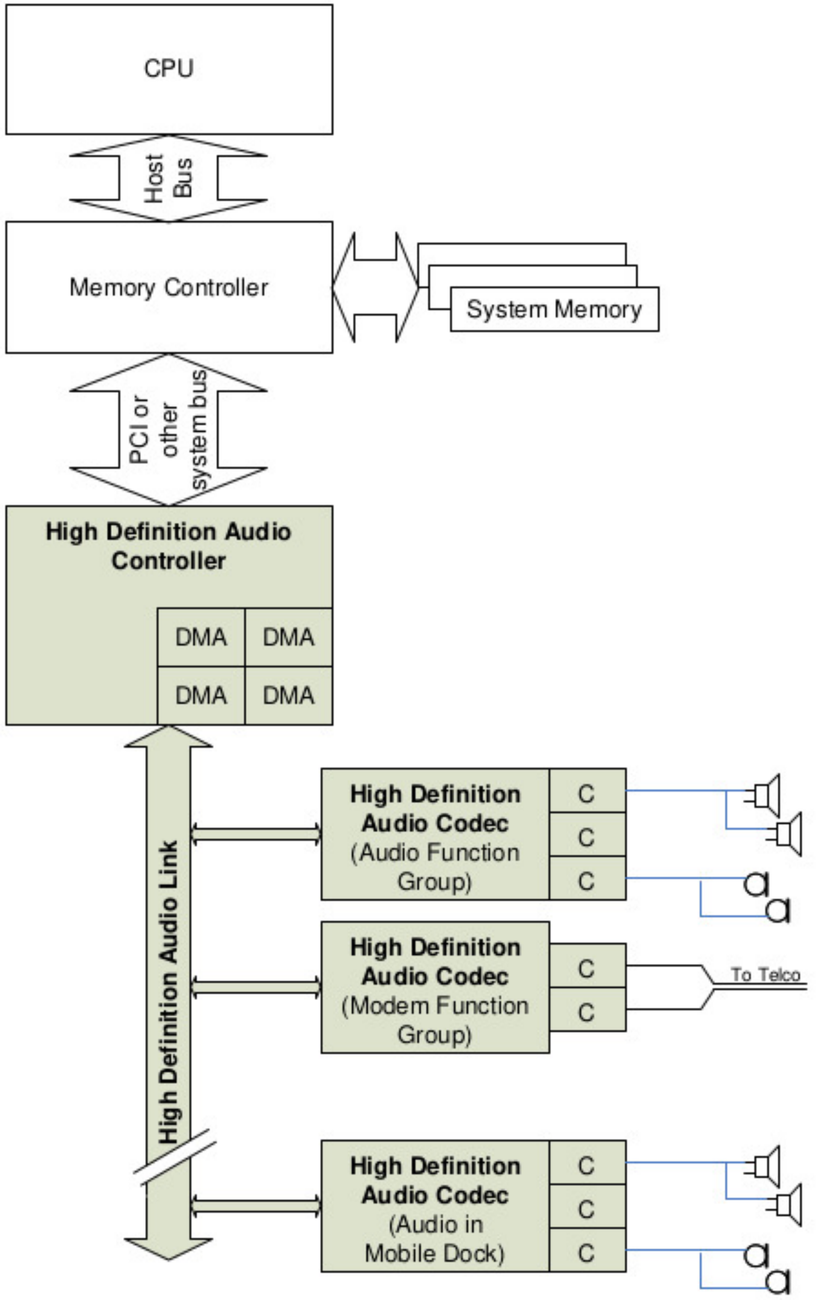
Intel HD Audio

Architektur

- Controller (Register, DMA)
  → Schnittstelle zum Rechner
  → Gehirn

- Link (physische Verbindung zwischen Controller und Codecs)
  → bidirektionaler Nervenstrang

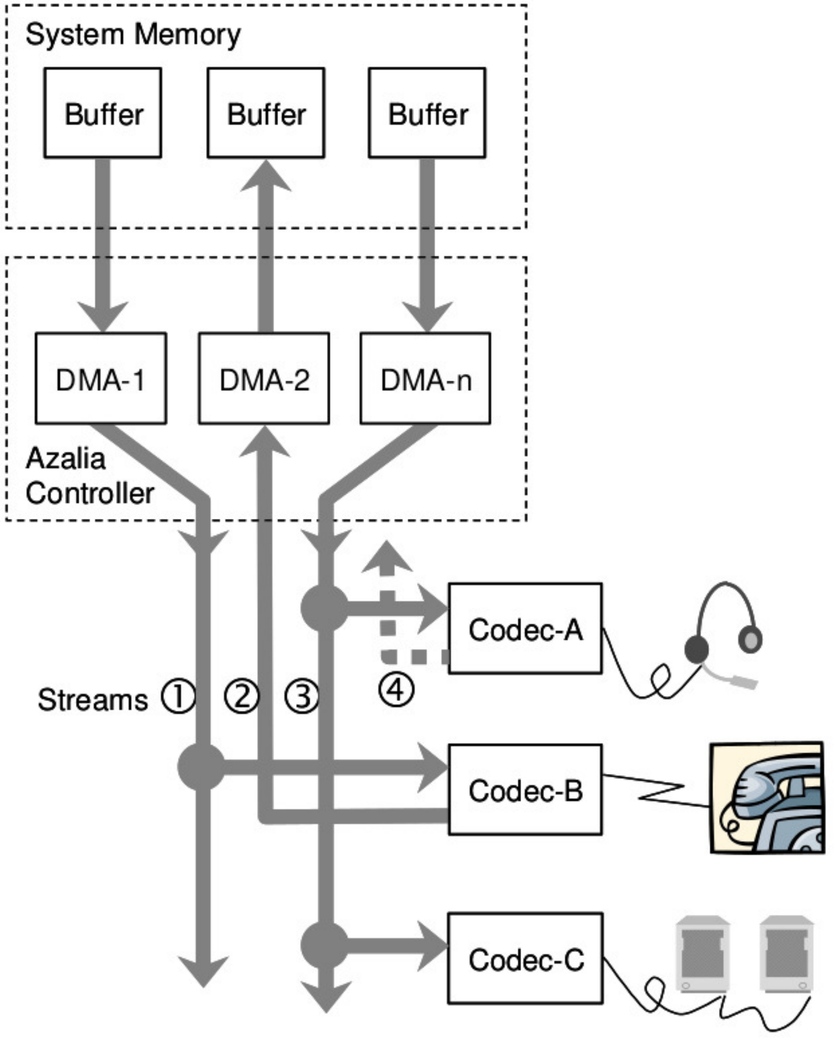
- Codec (DACs/ADCs)
  → Schnittstelle zur Außenwelt
  → Mund und Ohren



Intel HD Audio

Streams

- Stream: Verbindung zwischen mindestens zwei Audiobuffern und einem oder mehreren Codecs
- 1 Stream ↔ 1 DMA-Engine ↔ 1 Stream ID ↔ 1 Buffer Descriptor List
- entweder Input-Stream oder Output-Stream
- Stream Format (Samplerate, Bittiefe, bis zu 16 Channels)



Intel HD Audio

Programmiermodell

- Initialisierung des PCI Interfaces (Gerät finden, Bits im PCI Configuration Space setzen, Interrupt Line verbinden, Mapping der Register)
- Start und Konfigurierung des Controllers
- Ermittlung der Topologie des Codecs
- Erstellung eines Streams (Buffer allozieren, DMA-Engine über Register konfigurieren)
- Konfigurierung des Codecs für die Audiowiedergabe (inklusive Finden eines Ausgabepfads)
- Start des Streams

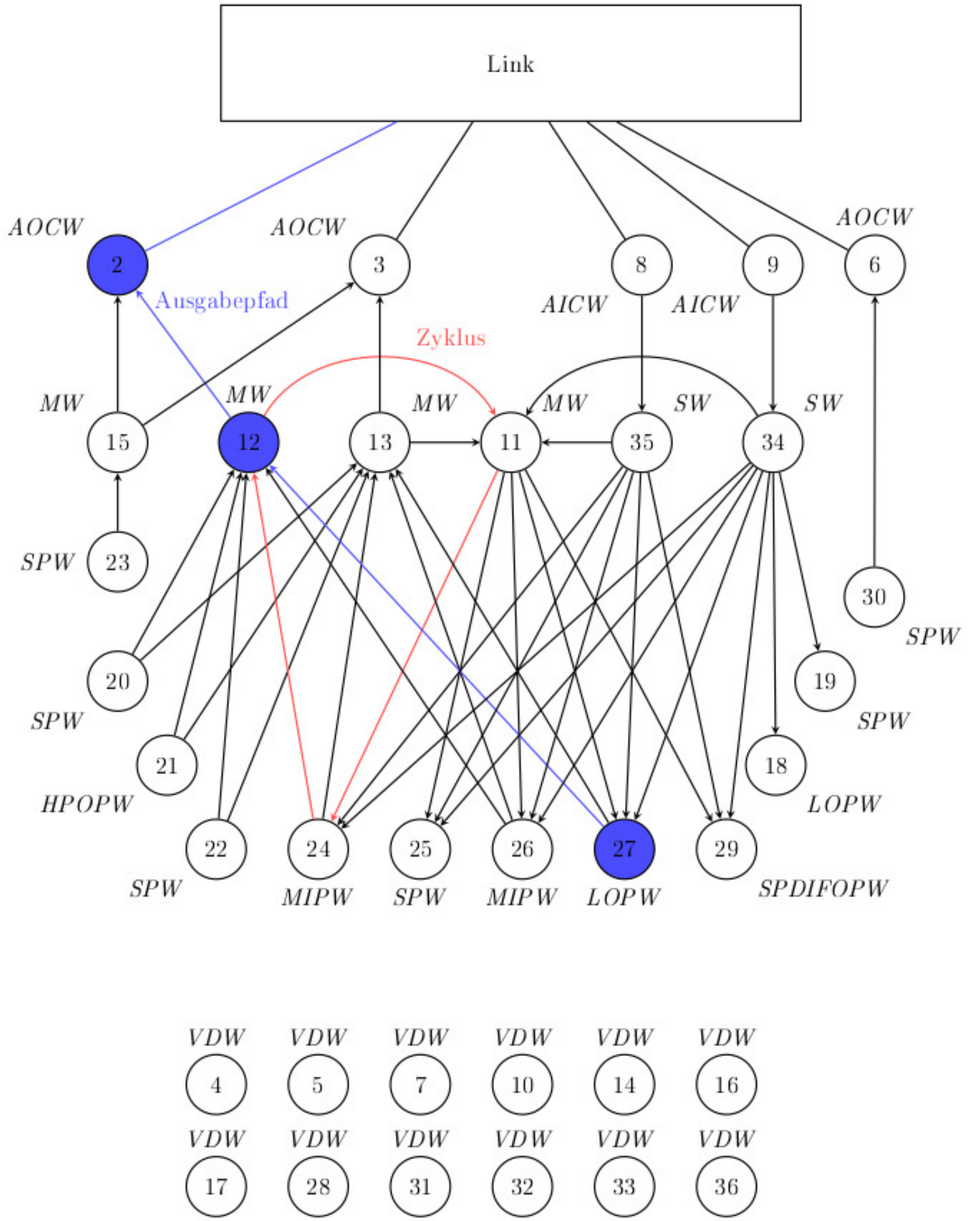


Abbildung 6.1: Verknüpfungsgraph der *Widgets* innerhalb des *Codecs* der in den Testrechner verbauten integrierten Soundkarte

Struktur des Treibers

- Objektorientierter Ansatz (Entitäten → Enums, Structs, Funktionalität → Methoden, Funktionen)
- 4 Module: API, PCI, Controller und Codec
- Offene Schichtenarchitektur:
  API → PCI, Controller, Codec
  Controller → Codec

Integration des Treibers in das D³OS

- Steuerung der Soundkarte über Methoden des im API-Modul definierten Structs IntelHDAudioDevice → API
- Treiberinitialisierung im Rahmen des Systemstarts
- Dort Aufruf des Konstruktors von IntelHDAudioDevice → Referenz auf erzeugte Instanz wird in statischer Variable gespeichert
- Auf Instanz kann von jedem Prozess im Kernel-Adressraum aus zugegriffen werden

Funktionalität des Treibers

- Benchmarks nicht sinnvoll, da Audio mit konstanter Datenrate übertragen wird
- Funktionierende Audiowiedergabe beweist implizit korrekte Umsetzung des Programmiermodells
- Für nicht für die Audiowiedergabe benötigte Buffer (CORB und RIRB, DMA Postition Buffer) wurden Testfunktionen geschrieben
- Aufruf der Demo-Funktion aus dem boot.rs heraus (also von außerhalb der Module des Treibers) zeigt, dass API funktioniert

Demo

- Output-Stream
- Samplerate: 48 kHz, Bittiefe: 16 Bit, Channel: 2 (stereo)
- 8 Audiobuffer mit je 64 * 4096 Byte
  → Abspieldauer pro Buffer: 64 * 4096 Byte / 48 kHz / 16 Bit / 2 ≈ 1.36533s
- In jedem Audiobuffer Sägezahnschwingungen in PCM-Darstellung, Frequenz verdoppelt sich jedes Mal: 50 Hz, 100 Hz, 200 Hz, 400 Hz, 800 Hz, 1600 Hz, 3200 Hz, 6400 Hz

Herausforderungen bei der Treiberentwicklung

- Komplexität und Offenheit der Intel HD Audio Architektur (wenige Sekundärquellen, modulare Codec-Architektur → kein universeller Algorithmus zur Konfigurierung eines Codecs → Treiber unterstützt momentan genau ein Modell)
- Testen auf virtualisierter Hardware in QEMU (Registerzugriffe funktionierten, aber Controller der virtualisierten Soundkarte interagierte nicht mit anderen DMA-Bereichen → Testen auf echter Hardware notwendig)
- Das noch junge D³OS (Betriebssystemfunktionalität für Konfigurierung des PCI-Interfaces und No-Cache-Allocation musste selbst geschrieben werden, Komplexität des D³OS konnte nur bedingt ausgeblendet werden)

Mögliche Erweiterungen

Fazit und Ausblick

- Interrupts
- Input-Streams
- CORB und RIRB statt Immediate Command Input and Output Registers nutzen
- Dateiformat für PCM-Daten
- Synchronisation (von Streams und Controllern), Striping, Power Management
- mehr Modelle unterstützen